Parallel multigrid methods : implementation on SUPRENUM-like architectures and applications

Karl Solchenbach, Clemens-August Thole, Ulrich Trottenberg

INRIA

UNITE DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France

Tel (1) 39 63 55 11

Rapports de Recherche

N° 746

PARALLEL MULTIGRID METHODS : IMPLEMENTATION ON SUPRENUM-LIKE ARCHITECTURES AND APPLICATIONS

Karl SOLCHENBACH
Clemens-August THOLE
Ulrich TROTTENBERG

NOVEMBRE 1987

Parallel Multigrid Methods: Implementation on SUPRENUM-Like Architectures and Applications

Karl Solchenbach
Clemens-August Thole
Ulrich Trottenberg

Méthodes multigrilles parallèles : implémentation sur des architectures de type SUPRENUM et applications

Karl Solchenbach
Clemens-August Thole
Ulrich Trottenberg

Suprenum GmbH, Hohe Str. 73, D-5300 Bonn
Gesellschaft für Mathematik und Datenverarbeitung (GMD),
D-5205 St. Augustin

Résumé

Les méthodes multigrilles de résolution d'équations aux dérivées partielles (et d'autres modèles mathématiques appliqués au calcul scientifique) se sont révélées être optimales sur architectures séquentielles. On cherche à les utiliser sur calculateurs parallèles ou vectoriels afin de tirer profit à la fois de leur haute efficacité (comparée aux méthodes classiques) et de toute la puissance de calcul de ces machines modernes. Il existe par conséquent un besoin en méthodes multigrilles parallèles. Certaines des méthodes maintenant bien connues (avec relaxation red-black et "zebra-type", voir [25]) présentent déjà un degré de parallélisme suffisamment élevé.

Parmi les architectures nouvelles de supercalculateurs, les systèmes MIMD à grand nombre de processeurs vectoriels et mémoires distribuées semblent promis à un bel avenir. La machine virtuelle SUPRENUM s'adapte de façon naturelle à ces architectures. Elle repose sur un modèle dynamique de processus, où chacun possède sa propre zone de données et échange des messages avec les autres.

Nous montrons dans cet article comment utiliser de tels architectures et concepts logiciels pour la résolution de gros problèmes avec maillage (équations aux dérivées partielles discrétisées, etc...). Le découpage en sous-domaines et la structuration par blocs - la communication ne se faisant qu'aux points frontières des sous-domaines ou des blocs - sont les approches naturelles dans ce cas. Les méthodes à base de maillages, en particulier les multigrilles, peuvent être ainsi efficacement parallélisées.

Dans le cadre du projet SUPRENUM ont été développés des utilitaires performants (comme par exemple des logiciels de visualisation : des processus/processeurs, des échanges de données entre grilles); ces programmes facilitent grandement l'implémentation de méthodes monogrilles ou multigrilles sur des systèmes multiprocesseurs à mémoires distribuées. Des codes multigrilles parallèles ont été exécutés sur le simulateur SUPRENUM [16], sur le "pré-prototype" SUPRENUM [22] et sur d'autres machines à mémoires distribuées telles l'Intel iPSC et l'hypercube CalTech.

Parallel Multigrid Methods: Implementation on SUPRENUM-Like Architectures and Applications †

Karl Solchenbach
Clemens-August Thole
Ulrich Trottenberg

Suprenum GmbH, Hohe Str. 73, D-5300 Bonn
Gesellschaft für Mathematik und Datenverarbeitung, D-5205 St. Augustin

Abstract

Multigrid (MG) methods for partial differential equations (and for other important mathematical models in scientific computing) have turned out to be optimal on sequential computers. Clearly, one wants to apply them also on vector and parallel computers in order to exploit both, the high MG-efficiency (compared to classical methods) and the full computational power of modern supercomputers. For this purpose, parallel MG methods are needed. It turns out that certain well-known standard MG methods (with RB and zebra-type relaxation, as described in [25]) already contain a sufficiently high degree of parallelism.

Among innovative supercomputer architectures, MIMD multiprocessor computers with local memory and a vector unit in each processor are particularly promising. A software approch that corresponds to such architectures in a natural way is the abstract SUPRENUM concept. It is characterized by a dynamical process system, where each process has its own data space and communicates with other processes by message-passing.

In this paper, we show how such architectures and software concepts are used for the solution of large scale grid problems (discrete PDEs, etc.). Grid partitioning and blockstructuring – with communication only along the subgrid or block boundaries – are the natural approches in this context. Any grid oriented method, in particularly any MG method can be efficiently parallelized using these approaches. In the SUPRENUM project, powerful software tools (e.g. a mapping library for the process-processor mapping and a communication library for the intergrid data exchanges) are developed that make it very easy to implement single grid and MG methods on local memory multiprocessor systems. Parallel MG programs have been run on the SUPRENUM simulator [16], the SUPRENUM pre-prototype [22] and some other local memory machines like the Intel iPSC and the CalTech hypercube.

†This report has been presented as an invited paper in the "International Conference on Supercomputing" in Athens, 8-13 June 1987. It was finished during a research stay of the third author at INRIA Centre de Sophia Antipolis (in connection with the 1987 Alexander-von-Humboldt award for the French-German scientific cooperation).

1 Parallel Multigrid

For a wide class of problems in scientific computing, in particular for partial differential equations, the multigrid (more general: the multi-level) principle has proved to yield highly efficient numerical methods [13,2,4,14,20]. However, the principle has to be applied carefully: if the "multigrid components" are not chosen adequately with respect to the given problem, the efficiency may be much smaller than possible. This has been demonstrated for many practical problems. Unfortunately, the general theories on multigrid convergence do not give much help in constructing really efficient multigrid algorithms. Although some progress has been made in bridging the gap between theory and practice during the last few years , there are still several theoretical approaches which are misleading rather than helpful with respect to the objective of real efficiency. The research in finding highly efficient algorithms for non-model applications therefore is still a sophisticated mixture of theoretical considerations, a transfer of experiences from model to real life problems and systematical experimental work. The emphasis of the practical research activity today lies – among others – in the following fields:

- finding efficient multigrid components for really complex problems, e.g. Navier-Stokes equations in general geometries
- combining the multigrid approach with advanced discretization techniques: using dynamic local multigrid refinements; adding artificial terms (viscosity, pressure, compressibility, etc.) in certain multigrid components; using "double" discretization, r-extrapolation, defect correction in connection with multigrid to obtain higher accuracy; using coarse-grid continuation techniques etc. [2]
- constructing highly parallel multigrid algorithms

In this paper, we want to deal only with the last topic.

Multigrid (MG) methods are known to be "optimal", i.e. the number of arithmetic operations that have to be performed is proportional to the number of discrete unknowns which are to be calculated. This statement directly applies to standard sequential MG algorithms. With the availability of parallel computers, the question arises how MG methods are suited for parallel computing. Sometimes one can find the conjecture that MG is – in some sense – an essentially sequential principle, or the opinion that the full MG efficiency

is obtained only on sequential computers and that there is always a loss of efficiency for MG on parallel architectures.

We do not intend to give a final answer to this question, but we want to contribute to a clarification of the situation.

First, one may distinguish the approaches where standard MG algorithms are discussed under the parallel aspect from those approaches where essentially new MG algorithms (or better: MG-like algorithms) are designed with respect to parallel computing. We will not discuss new algorithms of this latter type in this paper; we only want to make a few remarks about them.

The – in our opinion – most interesting proposal for such a variant has recently been made by Fredericson/McBryan [7]. Here on each level several coarse grid problems are solved simultaneously in order to improve the MG-convergence. This appraoch seems to be of considerable use particularly for massively parallel machines like the Connection Machine [18].

To the class of new MG-like algorithms belong also all those attempts where several levels are simultaneously employed. Such methods have been considered by Gannon/van Rosendale [8], Greenbaum [10], and others. A breakthrough has, however, not yet been achieved; for theoretical reasons, one may also doubt whether these approches can give a remarkable gain at all.

In this paper, we consider only standard MG methods under the parallel aspect. This means, in particular, that we regard the schedule according to which the different grid levels are passed through as essentially sequential. On each grid level, however, we perform each of the grid operations (the MG components: relaxation, computing of defects, interpolation, and restriction) as parallel as possible. It has been known for long that certain relaxation methods are parallel in a natural way, e.g.

- *Jacobi*-type relaxations

and

- *Gauss-Seidel*-type relaxations with *multi-color* (red-black, four color etc.) ordering of the grid points.

Clearly, also computing of defects, interpolation and restriction can be performed in parallel.

The first systematical papers on parallel MG were those of Grosch [11,12] and Brandt [3]. In [3] most of the essential phenomena with parallel MG are already discussed or at least mentioned. In particular, it is stated that the time complexity $T^*(N)$ (measured by the number of parallel arithmetic operations) of a suitable standard parallel full multigrid (FMG) solver for the 2D-Poisson model equation is $T^*(N) = O\,(log_2 N)^2$, where $N =$ number of grid points.

In this paper, we consider two model problems in some detail:

Example 1: A parallel MG-solver for the 3D-Poisson equation on the unit cube $(0,1)^3$ with periodic boundary conditions. A V-cycle of this algorithm is characterized by the following components (for a more detailed description see [26]).

Discrete operator:	ordinary second order 7-point approximation Δ_h on a regular cubic grid with meshsize h and $N = h^{-3}$
Relaxation:	3D-red-black pointwise, all red (black) grid points are simultaneoulsy treated in the first (second) relaxation half step; $\nu_1, \nu_2 = 1$ relaxation steps.
Coarsening:	standard coarsening $h \longrightarrow 2h$ ordinary 7-point operator Δ_{2h}
Grid transfers:	$h \longrightarrow 2h$: 3D full-weighting $2h \longrightarrow h$: trilinear interpolation
Cycle type:	V-cycle, correction scheme

The time complexity for a V-cycle of this algorithm is $T^*(N) = O(log_2 N)$.

Example 2: The 2D-Stokes equations

$$\begin{aligned} -\nabla^2 u + \frac{\delta p}{\delta x} &= f^1 \\ -\nabla^2 v + \frac{\delta p}{\delta y} &= f^2 \\ \frac{\delta u}{\delta x} + \frac{\delta v}{\delta y} &= f^3 \end{aligned}$$

defined on $\Omega = (0,1)^2$ with boundary conditions

$$\begin{aligned} u &= g^1 \\ v &= g^2 \end{aligned}$$

on $\partial\Omega$. In order to guarantee a unique solution the usual compatibilty condition is required additionally.

The Stokes equations are discretized in the usual way on a staggered quadratic grid (mesh-size h). p is defined in cell centers, whereas u and v are defined on the centers of the cell faces.

A V-cycle of a parallel MG-solver for this discrete problem is characterized by the following components (see also [21] for details):

Relaxation:	One relaxation step consists of two parts: Firstly, the momentum equations are relaxed for u and v simultaneously using fixed values of p. Then a so-called distributive relaxation sweep [2] is performed which updates the unknowns u, v and p in order to fulfil the continuity equation. Both parts of the relaxation are performed in a red-black ordering. Altogether, $\nu_1 = 1$, $\nu_2 = 2$ of these relaxation steps are carried out on each level.
Coarsening:	standard coarsening $h \longrightarrow 2h$ on staggered grids (the coarse grid is no subset of the fine grid) ordinary 7-point operator Δ_{2h}
Grid transfers:	$h \longrightarrow 2h$: 2D half-weighting on staggered grids $2h \longrightarrow h$: bilinear interpolation
Cycle type:	V-cycle, correction scheme

The time complexity of this algorithm (V-cycle) is also $T^*(N) = O(log_2 N)$.

In the two algorithms above only *pointwise* relaxation is needed for smoothing, since the corresponding equations are isotropic. In [27] a systematical first study for the practically important case of anisotropic 3D-operators has been presented. Here line and/or plane relaxation have been used for smoothing, and corresponding parallel algorithms have been described.

2 Parallel supercomputers, the SUPRENUM prototype

For the last decade, a lot of programs for large-scale scientific computing have been developed and run on vector supercomputers of the CRAY-1 class with great success. Today, the evolution of single processor vector machines seems to be stagnating and most of the increasing computational power is achieved by using more than one processor in parallel. Examples for this development are the CRAY-X/MP, CRAY-2/3, ETA^{10}, and several smaller systems, all with a small number of processors.

On the other hand, certain highly parallel computers with a large number of processors are entering the supercomputer market (Intel's iPSC-VX, FPS-T-Series, AMETEK S14, NCUBE, Connection Machine). Typically, these multiprocessor computers do *not* necessarily provide a common (shared) memory for all single processors. In this paper, we will concentrate our conciderations on local memory multiprocessor computers.

The hardware architecture of these multiprocessor computers with local memory is characterized by P processor "nodes", each of which has (at least)

- *its own CPU*
- *a floating point arithmetic*
- *a private local memory unit*
- *a communication component*

The nodes are connected by an *interconnection network for communication.* This network typically consists of a subset of all possible node connections (tree, grid, ring, hypercube, ...). Also any other topology based on buses may be used. – Each floating-point unit may be based on a scalar or vector arithmetic processor. – Usually the multiprocessor computer is connected to a front end computer for control (initialization, termination), I/O, and user interaction.

Typical numbers of P for specific machines are

$P \leq$	128	Intel iPSC
$P \leq$	64	Intel iPSC-VX
$P \leq$	256	AMETEK S14
$P \leq$	16K	FPS-T-Series (theoretically)
$P \leq$	1K	NCUBE
$P =$	256	SUPRENUM prototype

(Also some architectures which do not fulfil the characteristic criteria may be used like multiprocessor computers of the type above. For example, the $P = 64K$ processors of the Connection Machine CM-1 may be clustered to functional units which can perform floating-point arithmetic.)

Figure 1 shows the overall structure of the SUPRENUM-prototype hardware as it was designed by Giloi [9]. In the SUPRENUM-prototype 256 nodes are connected via a two-level interconnection network of buses.

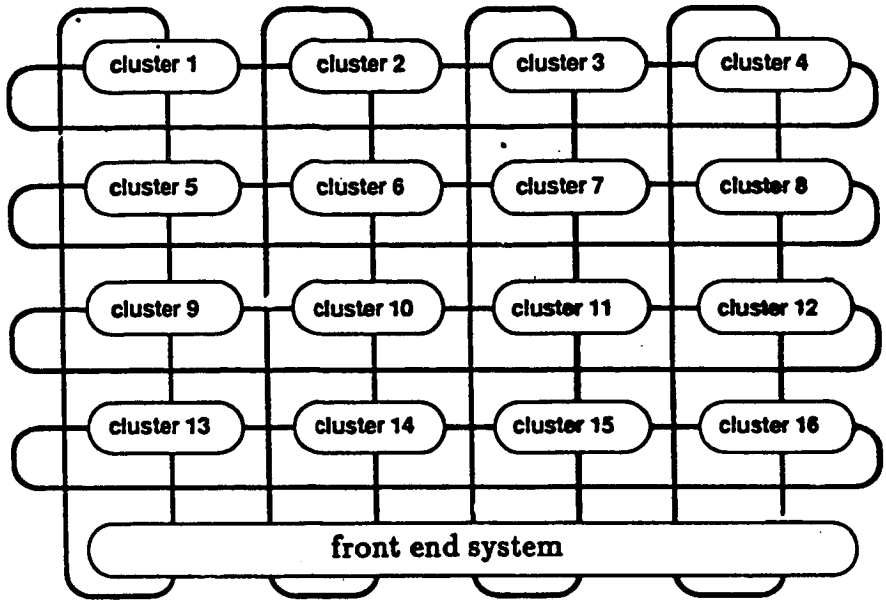

Figure 1: Structure of the SUPRENUM-prototype with 256 processors in 16 clusters

Each *node* consists of the MC68020 CPU, 8 Mbyte of private memory, a fast floating-point vector unit (8 Mflop/s peak performance, 16 Mflop/s with chaining) and dedicated communication hardware.

Up to 16 of these computing nodes are combined to a *cluster* using the *clusterbus* (256 Mbyte/s). Each cluster also contains a local disk (1 Gbyte), a disk controller node, a monitor node which supports performance measurements, a communication node for the connection to the second bus level (the SUPRENUMBUS), and a spare computing node for fault tolerance reasons.

As shown in Figure 1, 16 of these clusters are connected by a matrix of serial high-speed SUPRENUMBUSes (280 Mbit/s) and form the *high performance kernel*. The *SUPRENUM-prototype* is completed by a front end system which is used for operating and maintaining the high performance kernel as well as for software developement.

3 A software concept based on message-passing

For MIMD multiprocessor computers with local memory, a software concept has turned out to be suitable that is based on a *process* system and on a *message-passing communication* handling. The process concept for SUPRENUM (the so-called *abstract SUPRENUM architecture*) is a dynamic one which is characterized by the following elements:

- Processes are autonomous program units which run in parallel.
- Processes can terminate themselves and can create but not terminate other processes.
- Processes communicate only by exchange of messages, and no shared memory is available.
- Applications are started by one initial (or host) process typically running on the front end machine.
- In arithmetic expressions and communication instructions, array constructs are especially supported.
- The user defined process system is homogeneous and independent from the actual hardware configuration. The two-level architecture (cluster structure) is not reflected in the abstract SUPRENUM architecture and is completely transparent to the user. The processes are mapped to the clusters and nodes at run-time.

Figure 2 shows, that this abstract SUPRENUM architecture is the central model in the system software. The user should write his codes only in terms of processes.

The mapping of processes to nodes is supported by the *mapping-library*. It provides optimal mapping startegies for some standard process systems (like trees, rings, grids) and uses heuristical strategies for irregular process structures.

The SUPRENUM *operating sytem* consists of three components residing on the front end system, the cluster level and on the node level. The front end system is operated under UNIX V[1]. On the cluster level the operating system supports the local disk, the performance analysis and the connection between the two communication levels. In each node a small operating system (PEACE) is responsible for the process scheduling and the message handling.

[1] registered trademark of AT&T

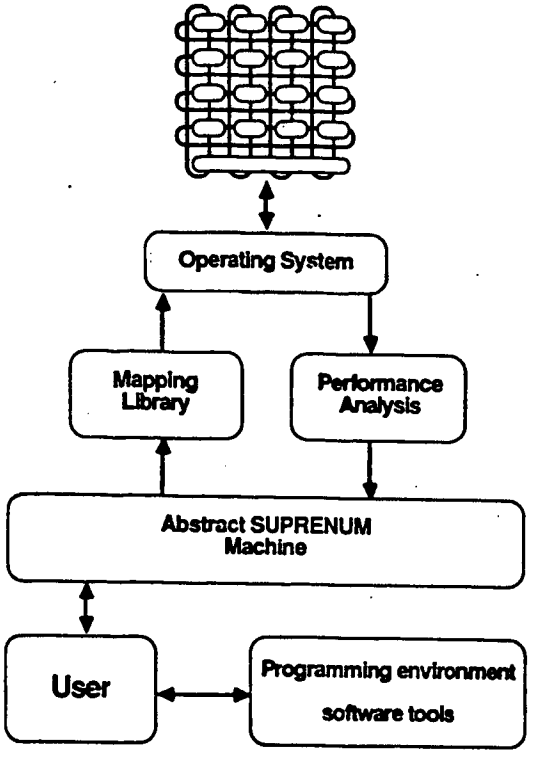

Figure 2: The SUPRENUM system software

The *programming language* for numerical computations is SUPRENUM-FORTRAN, an extended FORTRAN 77. The extensions include special process handling and message-passing constructs and an array syntax formulation according to the proposed FORTRAN-8X standard. Additionally Concurrent Modula-2 and a parallel version of C will be available.

Performance analysis tools will collect performance data from each cluster for graphical presentation. This enables the user to analyze the utilization of nodes and buses and to tune his parallel programs.

The SUPRENUM *programming environment* will provide a lot of tools which support the programmer in developing parallel software. Here, the syntax-directed editor, the communication library (see below), a SUPRENUM-simulator [16], an auto-vectorizer for SUPRENUM-FORTRAN and a visualization package for parallel program execution should be mentioned.

4 Grid partitioning

In Section 1 we have considered standard parallel MG algorithms which are highly parallel. However, if they are implemented on real vector or parallel computers, it usually is not possible to fully exploit their parallelism.

By the pipeline processors in vector computers for instance, only a low degree of parallelism can be achieved; and even highly parallel multiprocessor computers always have a certain limited number of simultaneoulsy working processing elements (say P). Nevertheless, the high degree of parallelism in the algorithms is useful or even necessary for several reasons. Firstly, it is preferable to construct algorithms which independently of P can be used on any parallel machine. Secondly, the full performance of vector units can usually be achieved the better the longer the occurring vectors are, i.e. the higher the degree of parallelism offered by the algorithm is. Finally, the recently designed high performance MIMD multiprocessor computers (like SUPRENUM) combine the – global – MIMD structure with – local – SIMD pipeline processing (vector floating point units) in each node. For such MIMD/SIMD systems, the MIMD and the SIMD degrees of parallelism are ***multiplied*** and have to be provided by the implemented algorithm.

We would like to emphasize that the communication problem in MIMD multiprocessor computers with local private memory has essential algorithmical implications. Since for such systems one has to make a decision about the interconnection structure of the nodes, this structure defines a "neighborhood" and by that, a topology of the nodes in a natural way. In the design of the algorithms this topology has to be taken into regard: Apart from the (sufficiently high degree of) parallelism that has to be provided by the algorithms, as a second important property "locality" of the algorithms of the algorithms with respect to the given topology is required. This means that the amount of data which have to be communicated, the number of communication packages, and the distances which have to be run through in the architecture become of essential significance.

If grid applications are to be implemented on MIMD multiprocessor computers, a straightforward approach is to use ***grid partitioning*** [26,19,23]. For all methods, single grid and MG, this means that the original domain is split into P parts (subdomains) in such a way that, with respect to the finest grid, each subdomain consists of (roughly) the same number of grid points (see Figure 3). Each subdomain is then assigned to one of the P processes of the parallel program. The partitioning generates certain artificial boundaries

within the original domain.

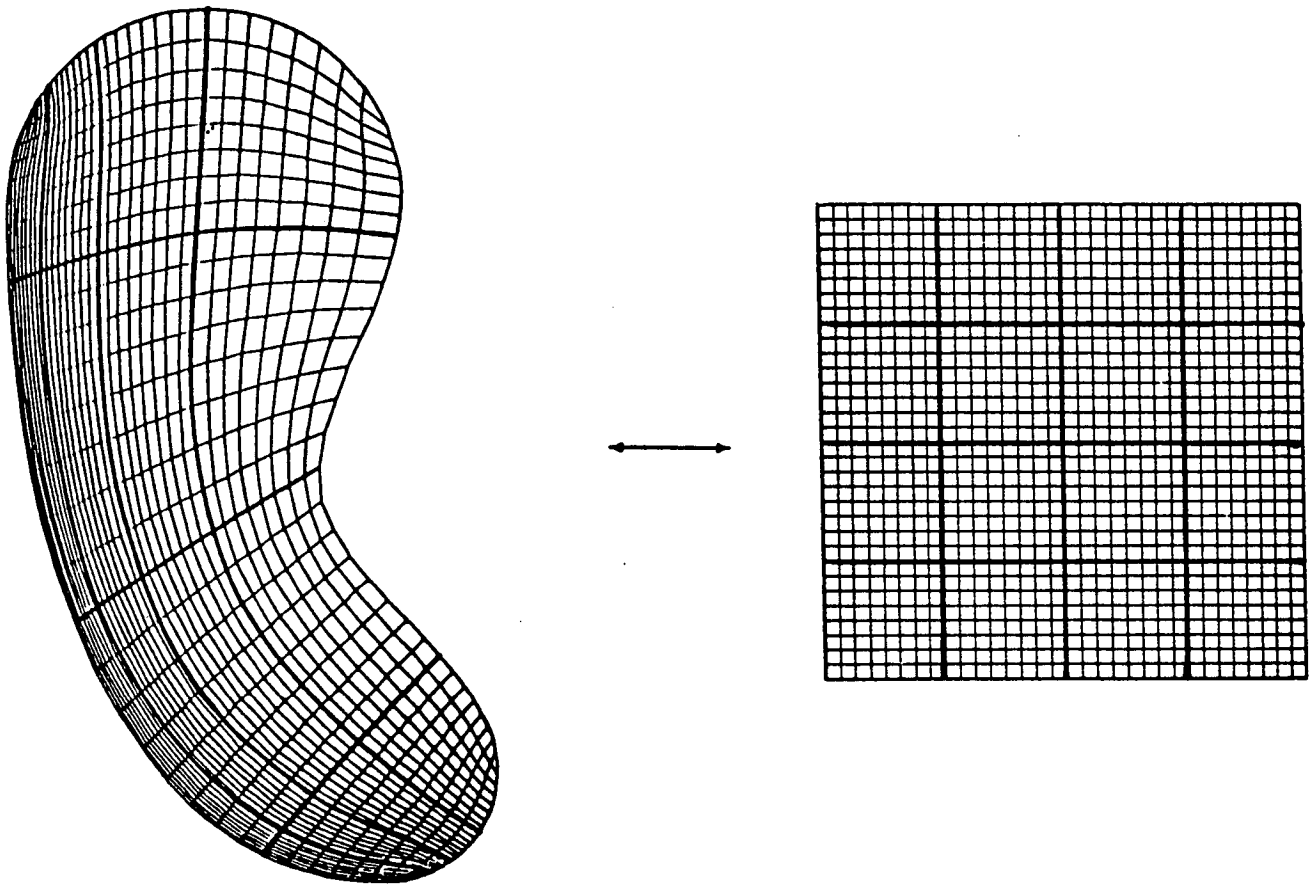

Figure 3: 2D-grid partitioned into 16 logically rectangular subdomains.

If we consider a typical component of a parallel grid algorithm, e.g. a parallel relaxation step, we see that on each subdomain this relaxation step can be carried out independently, provided all necessary data are available. Because of the only local dependencies of the grid points, each process needs foreign data only from boundary areas of neighboring subdomains. After the step is performed, again data have to be communicated (exchanged) along the artificial boundaries.

The extension of the single grid case to parallel MG is obvious: On the finest grid level, all communication is a strictly local one. Similarly, also on the coarser grids necessary communication is "local" relatively to the corresponding grid level (i.e. neighborhood is

defined with respect to the grid level).

One should distinguish the grid partitioning approach as sketched above from the decomposition [5] or substructuring methods [1] which are often considered in connection with finite element discretizations on parallel computers. The decomposition and substructuring methods lead to algorithms which are numercially different from the undecomposed or sequential version. In contrast to that, parallel algorithms based on grid partitioning are algorithmically equivalent to their non-partitioned versions (running on sequential computers) as the results of the partitioned and the non-partitioned versions are identical.

For the simple grid partitioning approach only static features of the process concept (as described in the previous section) are needed.

Parallel programs on SUPRENUM and on similar machines typically have the following structure:

- The host process creates the set of processes and sends them the necessary control data (identification of their "neighbors", index range of the subdomain, certain global parameters of the algorithm).
- The host process sends each process the initial data belonging to its part of the domain.
- The node processes receive the initial information.
- After each computational step the points near the interior boundaries (which are stored in *overlap areas*) are updated by mutual exchange of data.
- During the computation certain globally needed results (like norms of residuals) are assembled treewise.
- After the computation, the results are sent to the host process, where the solution for the entire domain is assembled.

For certain grid applications, the explicit programming of the communication can be hidden from the user. In the SUPRENUM project, for example, a library of communication routines has been developed [15] which ensures

- clean and error-free programming,

- easy development of parallel codes,
- portability within the class of local memory multiprocessor computers. Programs can be ported to any of these machines as soon as the communication library has been implemented.

Good experiences have been made in porting programs from the Intel iPSC to the SUPRENUM simulation system and vice versa. A corresponding library for block-structured (see below) applications is currently under development. Most of the application software, which is written in the SUPRENUM project, will be based on these routines.

5 Some multigrid results

If a certain program is implemented on the multiprocessor architecture the central question is how much faster the calculations can be executed here than on a single processor.

This is measured by the *multiprocessor speed-up*

$$S(N,P) = \frac{T(N,1)}{T(N,P)}$$

($T(N,P)$ = execution time for the parallel algorithm for a problem of size N on P nodes).

As usual, we furthermore define the *multiprocessor (MP)-efficiency*

$$E(N,P) = \frac{S(N,P)}{P}$$

Usually E is essentially < 1. Reasons for MP-efficiency losses are:

- The algorithm may not be totally parallelizable.
- The amount of work given to each processor is not balanced.
- The algorithm requires communication. (Start-up time for the initialization of communication is needed as well as transfer time for each item of the message.)
- Additional overhead is necessary for organization (handling of loops, general decisions, etc.).

(Note that the quantity E gives *no* information about the quality of an algorithm – its efficiency in the common sense – at all. It only says something about its parallelizability. A numerically totally inefficient algorithm may be optimally "MP-efficient".)

Example 1:

The 3D-Poisson MG-solver has been implemented on the CalTech Mark II hypercube [26]. Multiprocessor efficieny rates are given in Table 1. Obviously, the problem with $N = 8^3$ grid points is too small for a system with $P = 32$ nodes. However, already medium-sized problems with $N = 32^3$ grid points achieve an MP-efficiency of more than 50%.

N	time	$S(N,32)$	$E(N,32)$
8^3	0.306	6.8	0.21
16^3	0.847	12.4	0.39
32^3	3.370	18.6	0.58

Table 1: Computing time, MP-speed-up and MP-efficiency for a multigrid method for the 3D-Poisson equation with periodic boundary conditions using V-cycle with $\nu_1 = 2, \nu_2 = 1$ relaxations.

Table 2 gives a comparison of the MG-solver with a standard relaxation solver and with a FFT solver. With respect to MG, one has to be aware that on (very) coarse grids the following phenomena occur:

- The number of processors exceeds the number of grid points in some dimension. Some of the subdomains contain no grid point on coarse levels.
- The volume/surface ratio is getting smaller as the grids get coarser. On message passing parallel systems the communication time may dominate the computing time thus leading to inefficient algorithms.
- Since the messages become shorter on coarser grids the "start-up" time which is necessary to initialize exchange of surfaces of subgrids becomes more important.

These are the reasons for the worse MP-efficiency of MG algorithms. There are, of course, ways to come around this difficulty. The parallel MG implemetation (but not necessarily the parallel MG algorithm) which runs on the finer grids may be modified when the computation proceeds to the very coarse grids. A possible coarse grid strategy is:

Collect all coarse grid points to fewer and fewer processes as the coarsening proceeds. Although the numerical work per process is increased and unbalanced, a lot of communication time can be saved. This *agglomeration* strategy is included in the communication

library and causes no additional work for the user. Detailed investigations [24] show that – depending on the ratio of communciation versus calculation times – the correct coarse grid treatment may be crucial for the MP-efficiency.

method	time	$S(32^3, 32)$	$E(32^3, 32)$
relaxation	381.3	25.9	0.81
MG	3.4	18.6	0.58
FFT	22.0	29.8	0.93

Table 2: Computing time, MP-speed-up and MP-efficiency for different solvers applied to the 3D-Poisson equation with periodic boundary conditions and $N = 32^3$ grid points.

Although the MP-efficiency of the relaxation and the FFT solver is considerably better than for the MG-solver, the absolute computing time is essentially worse. So both, MP- *and* numerical efficiency are important in designing good algorithms for multiprocessor systems.

Example 2:

The 2D-Stokes MG solver (see Section 1) has been implemented on the Intel iPSC. Figure 4 shows the MP-efficiency rates in relation to the number of processors and the problem size. For a 64-hypercube, a problem size of $N = 256^2$ grid points is necessary in order to achieve an efficiency of more than 50%.

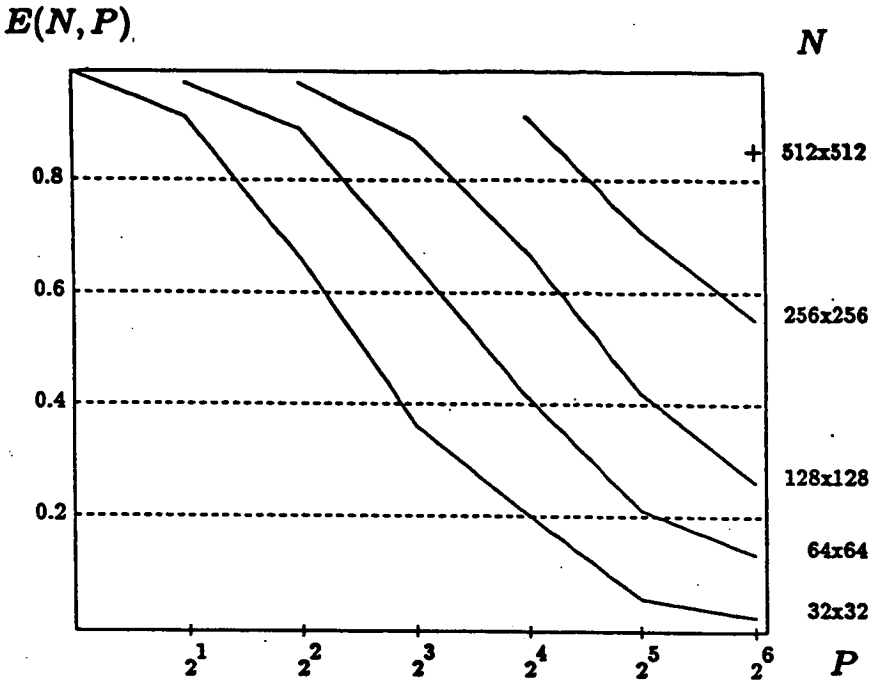

Figure 4: Parallel MG-code for the 2D-Stokes problem. Each curve shows the MP-efficiency for constant problem size N and increasing P.

6 Further applications in the SUPRENUM project

In the near future (up to 1988), the following CFD models will be implemented on the Suprenum computer.

- (Nonlinear) Potential equation for subsonic and transonic flow past airfoils. A Finite Volume (FV) discretization is used and a special relaxation is applied at grid points near the shock front (transonic case). This causes an inhomogeneous load distribution over the grid and special care has to be taken in order to avoid load unbalancing.

- Euler equations for 3D-computations employing advanced acceleration techniques. The system of equations is discretized by a FV scheme on a non-staggered grid. The stationary problem is solved by marching in pseudo time-steps (explicit Runge-Kutta method). The time marching can be viewed at as a Jacobi-type relaxation which is accelerated using coarser time-steps in a MG-like manner [28].

- Euler equations combined with boundary layer methods for the 2D- and 3D- simulation of flow past cars;
- Navier-Stokes equations (incompressible and compressible) for internal flows on general 2D- and 3D-domains. This is the most complex application imposing additional requirements on discretization, MG components and grid refinement strategies.
- Navier-Stokes (compressible) for the full simulation of the flow past cars (3D). This code allows also the simulation of large wake area which is impossible with Euler-/boundary layer methods.
- Grid generation codes for the generation of 2D- and 3D-boundary fitted grids. The grid structure is either a single logically rectangular grid (see Figure 3) or an arbitrary aggregation of blocks (=single grids). Single boundary fitted grids are generated by the solution of a system of Poisson-like equations (which corresponds to a mapping of the physical domain to a rectangular domain). This step itself may require large computing times and can be solved by MG methods [17]. We would like to point out that block structures have been introduced also independently from parallelization aspects for two reasons:
  - the geometry is too complex and cannot be mapped to a single rectangle (L-shaped domain)
  - due to core memory limits only a part of the grid can be computed at a time.

  In both cases the need to have regular data structures and to make efficient use of vector processing units motivates a partitioning of the grid into several logically rectangular blocks. Therefore existing codes which support block structures can easily be parallelized, in two steps:

  (1) by processing all blocks in parallel;

  (2) by parallelizing each block with respect to an overall balanced process size (load balancing).

In addition to the CFD codes, some selected simulation problems with huge computing time requirements will be implemented on Suprenum. These include particle simulation codes and nuclear reactor core simulations [6].

References

[1] Bjørstad, P.E., Widlund, O.B.: *Iterative methods for the solution of elliptic problems on regions partitioned into substructures.* SIAM J. Numer. Anal. 23, 6, 1986.

[2] Brandt, A.: *Multigrid techniques: 1984 guide with applications to fluid dynamics.* GMD-Studie No. 85, 1984.

[3] Brandt, A.: *Multigrid solvers on parallel computers.* In "Elliptic Problem solvers (M. Schultz, ed.)", Academic Press, New York, 1981.

[4] Braess, D., Hackbusch, W., Trottenberg, U. (eds.): *Advances in Multigrid methods.* Proceedings of the Conference Held in Oberwolfach, December, 8-13, 1984. Notes on Numerical Fluid Mechanics, Vol. 11, Vieweg, Braunschweig, 1984.

[5] Dihn, Q.V., Glowinski, R., Periaux, J.: *Solving elliptic problems by domain decomposition methods with applications.* In "Elliptic Problem Solvers II (G. Birkhoff and A. Schoenstadt, eds.)", Academic Press, New York, 1984, pp. 395-426.

[6] Finnemann H., Volkert, J.: *Parallel multigrid solvers for the neutron diffusion equation.* Proceedings of the International Topical Meeting on Advances in Reactor Physics, Mathematics and Computation, Paris, April, 27-30, 1987.

[7] Frederickson, P.O., McBryan, O.: *Parallel superconvergent multigrid.* Cornell Theory Center Technical Report CTC87TR12 7/87.

[8] Gannon, D.B., Rosendale, J,R. van: *Highly parallel multigrid solvers for elliptic PDEs: An experimental analysis.* Report 82-36, ICASE, NASA Langley Research Center, Hampton, VA, 1978.

[9] Giloi, W.K., Mühlenbein, H.: *Rationale and concepts for the Suprenum supercomputer architecture.* GMD, St. Augustin 1985.

[10] Greenbaum, A.: *A multigrid method for multiprocessors.* Appl. Math. Comp. 19, pp. 75–88, 1986.

[11] Grosch, C.E.: *Performance analysis of Poisson solvers on array computers.* Report TR 79-3, Old Domion University, Norfork, VA, 1979.

[12] Grosch, C.E.: *Poisson solvers on large array computer.* Proceedings 1978 LANL Workshop on vector and parallel processors (B.L. Buzbee and J.F. Morrison, eds.), 1978.

[13] Hackbusch, W., Trottenberg, U. (eds.): *Multigrid methods. Proceedings of the Conference held at Köln-Porz, November 23-27, 1981.* Lecture Notes in Mathematics Vol. 960, Springer, Berlin, 1982.

[14] Hackbusch, W., Trottenberg, U. (eds.): *Multigrid methods II. Proceedings of the 2nd Conference on Multigrid Methods, Cologne, Oct. 1-4, 1985.* Lecture Notes in Mathematics Vol. 1228, Springer, Berlin, 1986.

[15] Hempel, R., Schüller, A.: *Vereinheitlichung und Portabilität paralleler Anwendersoftware durch Verwendung einer Kommunikationsbibliothek.* Arbeitspapiere der GMD, Nr. 234, GMD, St. Augustin, 1986.

[16] Limburger, F., Scheidler, Ch., Tietz, Ch., Wessels, A.: *Benutzeranleitung des SUPRENUM-Simulationssystems SUSI.* GMD, St. Augustin, 1986.

[17] Linden, J., Stüben, K.: *Multigrid methods: An overview with emphasis on grid generation processes.* Arbeitspapiere der GMD Nr. 207, GMD, St. Augustin, 1986.

[18] McBryan, O.: *Numerical computation on massively parallel hypercubes.*, to appear.

[19] McBryan, O., Van de Velde, E.: *The multigrid method on parallel processors.* In [14].

[20] McCormick, S.F. (ed.): *Proceedings of the 2nd International Multigrid Conference, April 1985, Copper Mountain.* Appl. Math. Comp. Vol. 19, North Holland,1986.

[21] Niestegge, A., Stüben, K.: *A parallel multigrid method for the Stokes problem.* GMD-Arbeitspapier, GMD, St. Augustin , to appear.

[22] Peinze, K., Thole, C.A., Thomas, B., Werner, K.H.: *The SUPRENUM prototyping programme.* Suprenum-Report 5, SUPRENUM GmbH, Bonn, 1987.

[23] Rice, J.: *Parallel methods for PDEs.* Report CSD-TR-587, Purdue Univercity, West Lafayette, Indiana, 1986.

[24] Solchenbach, K.: *Parallel multigrid methods: Efficient coarse grid techniques.* Suprenum-Report, SUPRENUM GmbH, Bonn, to appear.

[25] Stüben, K., Trottenberg, U.: *Multigrid methods: Fundamental algorithms, model problem analysis and applications.* In [13]

[26] Thole, C.A.: *Experiments with multigrid methods on the CalTech-hypercube.* GMD-Studie Nr. 103, GMD, St. Augustin, 1985.

[27] Thole, C.A., Trottenberg, U.: *A short note on standard parallel multigrid algorithms for 3D-problems.* Suprenum-Report 3, SUPRENUM GmbH, Bonn, 1987.

[28] Wagner, B., Leicher, S., Schmidt, W.: *Applications of a multigrid finite volume method with Runge-Kutta time integration for solving the Euler and Navier-Stokes equations.* In GMD-Studie 110 (U. Trottenberg, W. Hackbusch, eds.), GMD, St. Augustin, 1986.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique